Complexity Theory

Lecture 10

Tom Gur

http://www.cl.cam.ac.uk/teaching/2324/Complexity

One Way Functions

A function f is called a *one way function* if it satisfies the following conditions:

1. f is one-to-one.

We cannot hope to prove the existence of one-way functions without at the same time proving $\mathrm{P} \neq \mathrm{NP}$.

It is strongly believed that the RSA function:

$$f(x, e, p, q) = (x^e \bmod pq, pq, e)$$

is a one-way function.

One Way Functions

A function f is called a *one way function* if it satisfies the following conditions:

1. f is one-to-one.
2. for each x, $|x|^{1/k} \leq |f(x)| \leq |x|^k$ for some k.

We cannot hope to prove the existence of one-way functions without at the same time proving $\mathsf{P} \neq \mathsf{NP}$.

It is strongly believed that the RSA function:

$$f(x, e, p, q) = (x^e \bmod pq, pq, e)$$

is a one-way function.

One Way Functions

A function f is called a *one way function* if it satisfies the following conditions:

1. f is one-to-one.
2. for each x, $|x|^{1/k} \leq |f(x)| \leq |x|^k$ for some k.
3. f is computable in polynomial time.

We cannot hope to prove the existence of one-way functions without at the same time proving $\mathsf{P} \neq \mathsf{NP}$.

It is strongly believed that the RSA function:

$$f(x, e, p, q) = (x^e \bmod pq, pq, e)$$

is a one-way function.

One Way Functions

A function f is called a *one way function* if it satisfies the following conditions:

1. f is one-to-one.
2. for each x, $|x|^{1/k} \leq |f(x)| \leq |x|^k$ for some k.
3. f is computable in polynomial time.
4. f^{-1} is *not* computable in polynomial time.

We cannot hope to prove the existence of one-way functions without at the same time proving $\mathrm{P} \neq \mathrm{NP}$.

It is strongly believed that the RSA function:

$$f(x, e, p, q) = (x^e \bmod pq, pq, e)$$

is a one-way function.

UP One-way Functions

We have

$$\mathsf{P} \subseteq \mathsf{UP} \subseteq \mathsf{NP}$$

UP One-way Functions

We have

$$\mathsf{P} \subseteq \mathsf{UP} \subseteq \mathsf{NP}$$

It seems unlikely that there are any NP-complete problems in UP.

UP One-way Functions

We have

$$\mathsf{P} \subseteq \mathsf{UP} \subseteq \mathsf{NP}$$

It seems unlikely that there are any NP-complete problems in UP.

One-way functions exist *if, and only if,* $\mathsf{P} \neq \mathsf{UP}$.

P $\neq$ UP Implies One-Way Functions Exist

Suppose that L is a language that is in UP but not in P. Let U be an *unambiguous* machine that accepts L.

$P \neq UP$ Implies One-Way Functions Exist

Suppose that L is a language that is in UP but not in P. Let U be an *unambiguous* machine that accepts L.

Define the function f_U by

if x is a string that encodes an accepting computation of U, then $f_U(x) = 1y$ where y is the input string accepted by this computation.

$f_U(x) = 0x$ *otherwise.*

We can prove that f_U is a one-way function.

Space Complexity

We've already seen the definition $\mathsf{SPACE}(f)$: the languages accepted by a machine which uses $O(f(n))$ tape cells on inputs of length n. *Counting only work space*.

Space Complexity

We've already seen the definition $\mathrm{SPACE}(f)$: the languages accepted by a machine which uses $O(f(n))$ tape cells on inputs of length n. *Counting only work space*.

$\mathrm{NSPACE}(f)$ is the class of languages accepted by a *nondeterministic* Turing machine using at most $O(f(n))$ work space.

Space Complexity

We've already seen the definition SPACE(f): the languages accepted by a machine which uses $O(f(n))$ tape cells on inputs of length n. *Counting only work space*.

NSPACE(f) is the class of languages accepted by a *nondeterministic* Turing machine using at most $O(f(n))$ work space.

As we are only counting work space, it makes sense to consider bounding functions f that are less than linear.

Classes

$\mathsf{L} = \mathsf{SPACE}(\log n)$

Classes

$\mathsf{L} = \mathsf{SPACE}(\log n)$

$\mathsf{NL} = \mathsf{NSPACE}(\log n)$

Classes

$\mathsf{L} = \mathsf{SPACE}(\log n)$

$\mathsf{NL} = \mathsf{NSPACE}(\log n)$

$\mathsf{PSPACE} = \bigcup_{k=1}^{\infty} \mathsf{SPACE}(n^k)$

The class of languages decidable in polynomial space.

Classes

$\mathrm{L} = \mathrm{SPACE}(\log n)$

$\mathrm{NL} = \mathrm{NSPACE}(\log n)$

$\mathrm{PSPACE} = \bigcup_{k=1}^{\infty} \mathrm{SPACE}(n^k)$

The class of languages decidable in polynomial space.

$\mathrm{NPSPACE} = \bigcup_{k=1}^{\infty} \mathrm{NSPACE}(n^k)$

Classes

$\mathsf{L} = \mathsf{SPACE}(\log n)$

$\mathsf{NL} = \mathsf{NSPACE}(\log n)$

$\mathsf{PSPACE} = \bigcup_{k=1}^{\infty} \mathsf{SPACE}(n^k)$

The class of languages decidable in polynomial space.

$\mathsf{NPSPACE} = \bigcup_{k=1}^{\infty} \mathsf{NSPACE}(n^k)$

Also, define:

Classes

$\mathsf{L} = \mathsf{SPACE}(\log n)$

$\mathsf{NL} = \mathsf{NSPACE}(\log n)$

$\mathsf{PSPACE} = \bigcup_{k=1}^{\infty} \mathsf{SPACE}(n^k)$

The class of languages decidable in polynomial space.

$\mathsf{NPSPACE} = \bigcup_{k=1}^{\infty} \mathsf{NSPACE}(n^k)$

Also, define:

co-NL – the languages whose complements are in NL.

Classes

$\mathsf{L} = \mathsf{SPACE}(\log n)$

$\mathsf{NL} = \mathsf{NSPACE}(\log n)$

$\mathsf{PSPACE} = \bigcup_{k=1}^{\infty} \mathsf{SPACE}(n^k)$

The class of languages decidable in polynomial space.

$\mathsf{NPSPACE} = \bigcup_{k=1}^{\infty} \mathsf{NSPACE}(n^k)$

Also, define:

co-NL – the languages whose complements are in NL.

co-NPSPACE – the languages whose complements are in NPSPACE.

Inclusions

We have the following inclusions:

$$\mathsf{L} \subseteq \mathsf{NL} \subseteq \mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PSPACE} \subseteq \mathsf{NPSPACE} \subseteq \mathsf{EXP}$$

where $\mathsf{EXP} = \bigcup_{k=1}^{\infty} \mathsf{TIME}(2^{n^k})$

Inclusions

We have the following inclusions:

$$\mathsf{L} \subseteq \mathsf{NL} \subseteq \mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PSPACE} \subseteq \mathsf{NPSPACE} \subseteq \mathsf{EXP}$$

where $\mathsf{EXP} = \bigcup_{k=1}^{\infty} \mathsf{TIME}(2^{n^k})$

Moreover,

$$\mathsf{L} \subseteq \mathsf{NL} \cap \text{co-}\mathsf{NL}$$

$$\mathsf{P} \subseteq \mathsf{NP} \cap \text{co-}\mathsf{NP}$$

$$\mathsf{PSPACE} \subseteq \mathsf{NPSPACE} \cap \text{co-}\mathsf{NPSPACE}$$

Padding arguments

We can scale up relations between complexity classes. For example:

$$\mathsf{L} = \mathsf{P} \implies \mathsf{PSPACE} = \mathsf{EXP}$$

Proof: Let $S \in \mathsf{EXP}$.

Padding arguments

We can scale up relations between complexity classes. For example:

$$\mathsf{L} = \mathsf{P} \implies \mathsf{PSPACE} = \mathsf{EXP}$$

Proof: Let $S \in \mathsf{EXP}$.

Then $S' = \{x 0 1^{2^{|x|^k}} \;:\; x \in S\} \in \mathsf{P}$.

Padding arguments

We can scale up relations between complexity classes. For example:

$$\mathsf{L} = \mathsf{P} \implies \mathsf{PSPACE} = \mathsf{EXP}$$

Proof: Let $S \in \mathsf{EXP}$.

Then $S' = \{x01^{2^{|x|^k}} \,:\, x \in S\} \in \mathsf{P}$.

Hence, $S' \in \mathsf{L}$.

Padding arguments

We can scale up relations between complexity classes. For example:

$$\mathsf{L} = \mathsf{P} \implies \mathsf{PSPACE} = \mathsf{EXP}$$

Proof: Let $S \in \mathsf{EXP}$.

Then $S' = \{x01^{2^{|x|^k}} : x \in S\} \in \mathsf{P}$.

Hence, $S' \in \mathsf{L}$.

Given $x \in S$, we can generate $x01^{2^{|x|^k}} \in S'$ in polynomial space.

Padding arguments

We can scale up relations between complexity classes. For example:

$$\mathsf{L} = \mathsf{P} \implies \mathsf{PSPACE} = \mathsf{EXP}$$

Proof: Let $S \in \mathsf{EXP}$.

Then $S' = \{x01^{2^{|x|^k}} : x \in S\} \in \mathsf{P}$.

Hence, $S' \in \mathsf{L}$.

Given $x \in S$, we can generate $x01^{2^{|x|^k}} \in S'$ in polynomial space.

Thus $S \in \mathsf{PSPACE}$.

Constructible Functions

Constructible Functions

A complexity class such as $\mathsf{TIME}(f)$ can be very unnatural, if f is.

We restrict our bounding functions f to be proper functions:

Constructible Functions

A complexity class such as $\mathsf{TIME}(f)$ can be very unnatural, if f is.

We restrict our bounding functions f to be proper functions:

Definition

A function $f : \mathbb{N} \to \mathbb{N}$ is *constructible* if:

- f is non-decreasing, i.e. $f(n+1) \geq f(n)$ for all n; and

Constructible Functions

A complexity class such as $\mathsf{TIME}(f)$ can be very unnatural, if f is.

We restrict our bounding functions f to be proper functions:

Definition

A function $f : \mathbb{N} \to \mathbb{N}$ is *constructible* if:

- f is non-decreasing, i.e. $f(n+1) \geq f(n)$ for all n; and
- there is a deterministic machine M which, on any input of length n, replaces the input with the string $0^{f(n)}$, and M runs in time $O(n + f(n))$ and uses $O(f(n))$ *work space*.

Examples

Examples

All of the following functions are constructible:

- $\lceil \log n \rceil$;

Examples

All of the following functions are constructible:

- $\lceil \log n \rceil$;
- n^2;

Examples

All of the following functions are constructible:

- $\lceil \log n \rceil$;
- n^2;
- n;

Examples

All of the following functions are constructible:

- $\lceil \log n \rceil$;
- n^2;
- n;
- 2^n.

Examples

All of the following functions are constructible:

- $\lceil \log n \rceil$;
- n^2;
- n;
- 2^n.

Examples

All of the following functions are constructible:

- $\lceil \log n \rceil$;
- n^2;
- n;
- 2^n.

If f and g are constructible functions, then so are $f+g$, $f \cdot g$, 2^f and $f(g)$ (this last, provided that $f(n) > n$).

Using Constructible Functions

NTIME(f) can be defined as the class of those languages L accepted by a *nondeterministic* Turing machine M, such that for every $x \in L$, there is an accepting computation of M on x of length at most $O(f(n))$.

Using Constructible Functions

NTIME(f) can be defined as the class of those languages L accepted by a *nondeterministic* Turing machine M, such that for every $x \in L$, there is an accepting computation of M on x of length at most $O(f(n))$.

If f is a constructible function then any language in NTIME(f) is accepted by a machine for which all computations are of length at most $O(f(n))$.

Using Constructible Functions

$\mathrm{NTIME}(f)$ can be defined as the class of those languages L accepted by a *nondeterministic* Turing machine M, such that for every $x \in L$, there is an accepting computation of M on x of length at most $O(f(n))$.

If f is a constructible function then any language in $\mathrm{NTIME}(f)$ is accepted by a machine for which all computations are of length at most $O(f(n))$.

Also, given a Turing machine M and a constructible function f, we can define a machine that simulates M for $f(n)$ steps.

Establishing Inclusions

To establish the known inclusions between the main complexity classes, we prove the following, for any constructible f.

Establishing Inclusions

To establish the known inclusions between the main complexity classes, we prove the following, for any constructible f.

- $\mathsf{SPACE}(f(n)) \subseteq \mathsf{NSPACE}(f(n))$;

Establishing Inclusions

To establish the known inclusions between the main complexity classes, we prove the following, for any constructible f.

- $\mathrm{SPACE}(f(n)) \subseteq \mathrm{NSPACE}(f(n))$;
- $\mathrm{TIME}(f(n)) \subseteq \mathrm{NTIME}(f(n))$;

Establishing Inclusions

To establish the known inclusions between the main complexity classes, we prove the following, for any constructible f.

- $\mathrm{SPACE}(f(n)) \subseteq \mathrm{NSPACE}(f(n))$;
- $\mathrm{TIME}(f(n)) \subseteq \mathrm{NTIME}(f(n))$;
- $\mathrm{NTIME}(f(n)) \subseteq \mathrm{SPACE}(f(n))$;

Establishing Inclusions

To establish the known inclusions between the main complexity classes, we prove the following, for any constructible f.

- $\mathrm{SPACE}(f(n)) \subseteq \mathrm{NSPACE}(f(n))$;
- $\mathrm{TIME}(f(n)) \subseteq \mathrm{NTIME}(f(n))$;
- $\mathrm{NTIME}(f(n)) \subseteq \mathrm{SPACE}(f(n))$;
- $\mathrm{NSPACE}(f(n)) \subseteq \mathrm{TIME}(k^{\log n + f(n)})$;

Establishing Inclusions

To establish the known inclusions between the main complexity classes, we prove the following, for any constructible f.

- $\mathrm{SPACE}(f(n)) \subseteq \mathrm{NSPACE}(f(n))$;
- $\mathrm{TIME}(f(n)) \subseteq \mathrm{NTIME}(f(n))$;
- $\mathrm{NTIME}(f(n)) \subseteq \mathrm{SPACE}(f(n))$;
- $\mathrm{NSPACE}(f(n)) \subseteq \mathrm{TIME}(k^{\log n + f(n)})$;

Establishing Inclusions

To establish the known inclusions between the main complexity classes, we prove the following, for any constructible f.

- $\mathrm{SPACE}(f(n)) \subseteq \mathrm{NSPACE}(f(n))$;
- $\mathrm{TIME}(f(n)) \subseteq \mathrm{NTIME}(f(n))$;
- $\mathrm{NTIME}(f(n)) \subseteq \mathrm{SPACE}(f(n))$;
- $\mathrm{NSPACE}(f(n)) \subseteq \mathrm{TIME}(k^{\log n + f(n)})$;

The first two are straightforward from definitions.

The third is an easy simulation.

The last requires some more work.

Reachability

Recall the Reachability problem: given a *directed* graph $G = (V, E)$ and two nodes $a, b \in V$, determine whether there is a path from a to b in G.

Reachability

Recall the Reachability problem: given a *directed* graph $G = (V, E)$ and two nodes $a, b \in V$, determine whether there is a path from a to b in G.

A simple search algorithm solves it:

1. mark node a, leaving other nodes unmarked, and initialise set S to $\{a\}$;

Reachability

Recall the Reachability problem: given a *directed* graph $G = (V, E)$ and two nodes $a, b \in V$, determine whether there is a path from a to b in G.

A simple search algorithm solves it:

1. mark node a, leaving other nodes unmarked, and initialise set S to $\{a\}$;
2. while S is not empty, choose node i in S: remove i from S and for all j such that there is an edge (i, j) and j is unmarked, mark j and add j to S;

Reachability

Recall the Reachability problem: given a *directed* graph $G = (V, E)$ and two nodes $a, b \in V$, determine whether there is a path from a to b in G.

A simple search algorithm solves it:

1. mark node a, leaving other nodes unmarked, and initialise set S to $\{a\}$;
2. while S is not empty, choose node i in S: remove i from S and for all j such that there is an edge (i, j) and j is unmarked, mark j and add j to S;
3. if b is marked, accept else reject.

We can use the $O(n^2)$ algorithm for Reachability to show that:

$\mathsf{NSPACE}(f(n)) \subseteq \mathsf{TIME}(k^{\log n + f(n)})$

for some constant k.

We can use the $O(n^2)$ algorithm for Reachability to show that:

$\mathsf{NSPACE}(f(n)) \subseteq \mathsf{TIME}(k^{\log n + f(n)})$

for some constant k.

Let M be a nondeterministic machine working in space bounds $f(n)$.

For any input x of length n, there is a constant c (depending on the number of states and alphabet of M) such that the total number of possible configurations of M within space bounds $f(n)$ is bounded by $n \cdot c^{f(n)}$.

> *Here, $c^{f(n)}$ represents the number of different possible contents of the work space, and n different head positions on the input.*

Questions?